REMARKS

In response to the Office Action dated August 26, 2004, Applicant provides the following comments. Claims 1-3, 7-20, 22 and 25-39 remain pending in the Application. No amendments to the claims have been made. Reconsideration of the Application is respectfully requested in view of the following comments.

The Official Action rejected Claims 1-3, 7, 8, 16-20, 22 and 32 under 35 U.S.C. §102(e) as being anticipated by *Hawkins*. Applicant respectfully traverses. Claim 1 as previously amended recites "wherein a non-core service is responsible for providing a functionality of the application and corresponds to a user interaction with the application during run-time, and wherein, in response to the user interaction during run-time, corresponding non-core services are designated a top priority in the priority order such that functionality of the application is enabled." The Official Action states that the language "wherein a non-core service is responsible for providing a functionality application and corresponds to a user interaction with the application during run-time" is disclosed at column 3, lines 57-65 and column 4, lines 23-39. Applicant does not believe that column 3, lines 57-65 and column 4, lines 23-39 describe a non-core service as providing a functionality that corresponds to a user interaction with the application during run-time. Applicant believes that the Official Action has misstated what is taught at column 3, line 57-65. This section is describing the operation of a tool for generating "archive" or "JAR" files for a program. The tool first analyzes the code of a program to determine splits or logical breaks in the program. This process may be done manually or using tracing techniques. The tool described in the *Hawkins* reference looks at a set of test cases and determines time periods when classes are not being instantiated. This analysis is used to divide the various classes into archive files. Thus, the disclosure in the *Hawkins* reference does not describe analyzing user interactions with a program during run-time. Instead, this reference describes analyzing the program flow of the application to determine logical breaks in the program where classes are not being instantiated.

Furthermore, the discussion in column 4, line 23-39 does not describe determining the order in which classes or services are required for providing functionality corresponding to user interaction. This section describes that application use classes may be divided into particular JAR files based upon the natural breaks that occur in the flow of execution of the code of a program being tested by the tool. The tool described in the *Hawkins* reference requires a number of initial tracing tests to be run upon a particular program to determine natural breaks in the program flow of the application. Using the information provided by these multiple previous tests, the classes are divided into particular archive or JAR files. These archive or JAR files are created based on the earlier analysis and are downloaded during break periods in an application flow. The division of classes into JAR files is not based upon user interactions with the application during run-time. In the tool described in the *Hawkins*, user interactions have no effect on the downloading of classes during run-time. The point at which classes are downloaded has been predetermined based upon the way the classes were grouped in the JAR files based upon the previous analysis. The non-core services of Claim 1 correspond to a user interaction with the application during run-time. The classes described in the *Hawkins* reference are grouped based upon program flow analysis that has occurred prior to run-time during the tracing process.

The Official Action further indicated that the designation of non-core services as a top priority in the priority order in response to user interaction during run-time was disclosed by FIGURE 3 and column 5, lines 17-32. Applicant has reviewed these sections and respectfully disagrees. FIGURE 3 merely indicates the particular peaks and valleys that occur for the use of classes during a program execution as discussed previously. Use Case 1 in FIGURE 3 indicates that during execution of the program, Class I and Class 2 are initially used, then there is a period of inactivity, then Class 3 is used, then there is another period of inactivity, then Class 4 is used, then there is a final period of inactivity, and then Class 5 is used. Likewise, Use Case 2 describes that Class 1 and Class 2 are used and then there is a period of inactivity, Class 3 is used and then there is a period of inactivity, Class 4 and Class 7 is used and there is a period of inactivity, and then Class 6 is used. The discussion with respect to FIGURE 3 merely discloses how a number of use cases, i.e., the tracings discussed previously, must be analyzed to determine the groupings of classes within particular JAR files. Once this analysis has been performed, and the particular groupings of classes within JAR files has been determined, the grouping

of classes is fixed. There is no designation of a particular non-core service being designated as a top priority in the priority order such that the functionality application is enabled in response to a user inaction during run-time. The designation of a particular class as top priority in the priority order is based upon the JAR file order created prior to execution of the program during the analysis phase.

The Applicant's Claim 1 describes a software engine which is much more robust in that any non-core service may be designated as top priority during run-time merely in response to the user interaction during run-time. Thus, a much more dynamic system is provided. The tool described in the *Hawkins* reference describes a different method for improving program operation that pre-analyzes the operation of the program and groups classes in an optimal fashion. Thus, the *Hawkins* reference uses analysis to determine a one size fits all designation for the order and groupings in which classes are downloaded for instantiation. In Hawkins, once the tool has determined an order, nothing can change the order when the application executes the application. Applicant's Claim 1 describes designating a particular non-core service as top priority based upon a user interaction during run-time. Thus, Applicant's claim 1 describes a software engine providing a much more robust and dynamic operating environment than that described by the system in the *Hawkins* reference.

The *Hawkins* disclosure is about determining, based on a collection of test cases, the best way to segment an application into groups of classes that are going to constitute archives. As explained in column 2 lines 23-27, once the list of archives has been determined, the VM loads them in their order of appearance in the Applet tag in the HTML file. Once the *Hawkins* tool finishes execution and determines a list of archive classes, the order of loading for the archives is fixed and cannot change dynamically when the application executes. The present invention does not teach how to group the functionality in groups of classes, it assumes that this work has been done by the developer or architect prior to deploying the application. *Hawkins* might be used to determine this grouping of classes into jar files. The present invention teaches a method of loading of classes at runtime, based on user interaction during runtime.

Therefore, Applicant believes *Hawkins* is distinguished from Claim 1 in a number of ways. The *Hawkins* reference is not concerned with services but merely grouping classes into groups for transmission to the client in an optimum manner. *Hawkins* does not teach or suggest a service as defined in Applicant's Claim 1. *Hawkins* does not teach or suggest an engine responsive to user functionality or user action which causes require additional services to be downloaded in order to execute subsequent functionality of an application. The present invention is responsive to the user needs and interaction during run-time, an advantage which *Hawkins* does not enjoy. The *Hawkins* reference merely attempts to simulate test cases for the application for all users or groups of users and then establishes a fixed, optimal archive file structure. The present invention takes a very different approach and adapts to individual user interaction with an application during run-time. Instead of determining the most optimal packaging of classes, it acknowledges the fact that a program is made of multiple services that can be triggered by user interaction and adapts the loading of those services based on the runtime behavior of the user. Therefore, the Applicant respectfully submits that Claim 1, and all claims dependent therefrom, is distinguishable from the art of record and a Notice of Allowance is respectfully requested.

Claim 17 includes limitations similar to those of previously discussed Claim 1. Applicant respectfully submits that Claim 17, and all claims dependent therefrom, are allowable over the *Hawkins* reference for similar reasons.

Claims 9-15, 25-31 and 33-39 were rejected under 35 U.S.C. §103(a) as being unpatentable over *Hawkins* as applied to claims 1 and 7 in view of *Judge, et al.* Applicant respectfully traverses. Applicant respectfully submits that each of the above-described claims, being dependent upon previously discussed claims 1 and 17 respectively, are allowable for similar reasons as the *Judge* reference fails to overcome the shortcomings of *Hawkins*. A *Notice of Allowance* is respectfully requested.

Applicant has now made an earnest attempt in order to place this case in condition for allowance. For the reasons stated above, Applicant respectfully requests full allowance of the claims as amended. Please charge any additional fees or deficiencies in fees or credit any overpayment to Deposit Account No. 20-0780/NEXU-26,961 of HOWISON & ARNOTT, L.L.P.

Respectfully submitted,
HOWISON & ARNOTT, L.L.P.
Attorneys for Applicant

Brian D. Walker
Registration No. 37,751

BDW/yoc

P.O. Box 741715
Dallas, Texas 75374-1715
Tel: 972-479-0462
Fax: 972-479-0464
November 24, 2004